惰性载入：函数执行的分支只会发生一次。（能力检测只检测一次，不用每次调用都检测）

函数绑定：bind（）方法，传入作为this值的对象。

函数柯里化：用于创建已经设置好了一个或多个参数的函数。

函数节流：常用于onresize，设置定时器延迟执行。

用于构建页面的语义元素

article\aside\figcaption\figure\footer\header\nav\section\details\summary

figure定义媒介内容的分组，以及它们的标题。（添加插图）

article文章

aside标签定义 article 以外的内容。aside 的内容应该与 article 的内容相关。

hgroup定义有关文档中的 section 的信息。

用于标识文本的语义元素

mark定义有记号的文本。

time

wbr Word Break Opportunity (<wbr>) 规定在文本中的何处适合添加换行符。

Web表单及交互

input(增加了很多类型)\datalist\keygen\meter\progress\command\menu\output

<form action=”” method=”GET/POST”>

<fieldset>

<legend>健康信息</legend>

<label for=”height”>身高：</label>

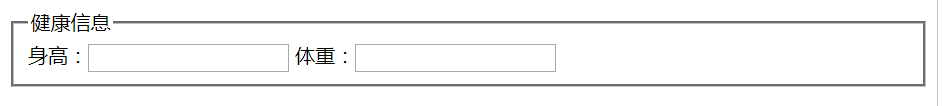
<input type="text" id=”height”>

<label for=”weight”>体重：</label>

<input type="text" id=”weight”>

</fieldset>

</form>



音频、视频及插件

audio\video\source\embed

Canvas

非英语支持

bdo\rp\rt\ruby

微数据

itemprop是属性名

itemscope属性表示要使用新词汇表提供属性数据

itemtype属性是以XML命名空间表示的词汇表（<http://schema.org/Person>）

表单

placeholder=“”占位符

autofocus自动获取焦点只能给一个input或textarea添加这个属性

required不能留空白

例子<input id=”name” placeholder=”John” autofocus required>

关闭表单验证：

1.在<form>中添加novalidate属性

2.添加一个额外按钮，在相应input中添加formnovalidate属性

验证样式挂钩（css伪类）：

required（必填）和optional（选填）：根据字段中是否使用了required属性来应用不同的样式

valid（有效）和invalid（无效）：根据控件中是否包含错误来应用不同的样式。（非实时提示）

in-range（在范围内）和out-of-range（超出范围）：根据空间的min和max来判断

focus伪类

要保证没有这些伪类页面也没问题（旧浏览器不支持）

例子：

input:required{

background-color:lightyellow；

}

input:required:invalid{

background-color:lightyellow;

}

pattern正则表达式

自定义验证setCustomValidity方法

<textarea id=”t” oninput=”validateComments(this)”></textarea>

至少20字符:

function validateComments(input){

if(input.value.length<20){

input.setCustomValidity(“you need to comment in more details.”);

}else{

input.setCustomValidity(“”);

}

}

input（number）

<input type="number" min="0" max="100" step="0.1" value="80">

input（range）滑动条

<input type="number" min="0" max="100" value="80">

input

date:2017-07-07

time:14:35或14:35:50.2

datetime-local:2017-05-05T18:30

month:2017年11月

week：2017年25周

color：chrome支持

datalist:

<input list="dd">

<datalist id="dd">

<option value="111">

<option value="121">

<option value="1311">

<option value="1151">

<option value="1141">

<option value="1161">

</datalist>

column-count:多栏文字（竖着排版）

column-width:每一栏有多宽（1em=2倍字体大小）

column-gap:调整分栏之间的间隔

column-rule:1px solid red;添加一条垂直的分割线

column-span:让图片和其他元素横跨多列，默认值1

web存储（数据自动转成字符串）

本地存储：localStorage对象

保存数据，使用localStorage.setItem()方法

如：localStorage.setItem(“user\_name”,”张三”);

或者保存文本框的值：

var nameInput=document.getElementById(“user\_name”);

localStorage.setItem(“user\_name”,nameInput.value);

获取数据，使用localStorage.getItem()方法

如：alert(“你的名字是”+localStorage.getItem(“user\_name”));

会话存储：sessionStorage对象

还可以使用索引或属性名方式读写数据（.和[]）

删除数据： removeItem()方法

localStorage.removeItem(“user\_name”);

清空网站在本地保存的会话数据

sessionStorage.clear();

查找所有数据项(得到属性名)：key()方法

for(var i=0;i<a.length;i++){

var key=localStorage.key(i);

}

数字：number（）

日期：先按照既定的格式把日期换成相应的文本，然后再根据取得的文本创建日期对象

window.onload=function () {  
 var today=new Date();  
 var todayString=today.getFullYear()+"/"+today.getMonth()+"/"+today.getDate();  
 sessionStorage.setItem("session\_started",todayString);  
  
 today=new Date(sessionStorage.getItem("session\_started"));  
 alert(today.getDate());  
};

onStorage事件

在一个域下的其中一个页面改变了本地存储或会话存储，那么这个域下的其他页面会触发onStorage事件（对存储不产生任何影响的操作不触发）

IndexedDB

离线应用

Ajax

function loadXMLDoc()

{

var xmlhttp;

if (window.XMLHttpRequest)

{

// IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码

xmlhttp=new XMLHttpRequest();

}

else

{

// IE6, IE5 浏览器执行代码

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");

}

xmlhttp.onreadystatechange=function()

{

if (xmlhttp.readyState==4 && xmlhttp.status==200)

{

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

}

}

xmlhttp.open("GET","/try/ajax/ajax\_info.txt",true);

xmlhttp.send();

}

服务器发送事件

消息格式：

以data:开头，然后是实际消息的内容，再加上换行符

data: The server has sent you this message.\n\n

还可以发送多行消息，每行要跟一个行结束符

data:hello\n

data:world\n\n

先创建一个EventSourse对象，传入服务器端发送消息的脚本URL

source=new EventSource(“qwe.php”);

将处理函数指定给onMessage事件，这个事件会在页面接受到消息后触发

source.onMessage=handle;

触发了handle函数后，可以从事件对象的data属性中取得消息

function handle(e){

div.innerHTML=e.data;}

最后调用EventSource对象的close（）方法，可以让页面停止监听服务器事件

source.close();

4.box-shadow

|  |  |  |
| --- | --- | --- |
| 值 | 描述 | 测试 |
| *h-shadow* | 必需。水平阴影的位置。允许负值。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_box-shadow) |
| *v-shadow* | 必需。垂直阴影的位置。允许负值。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_box-shadow) |
| *blur* | 可选。模糊距离。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_box-shadow&p=3) |
| *spread* | 可选。阴影的尺寸。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_box-shadow&p=7) |
| *color* | 可选。阴影的颜色。请参阅 CSS 颜色值。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_box-shadow&p=10) |
| inset | 可选。将外部阴影 (outset) 改为内部阴影。 | [测试](http://www.w3school.com.cn/tiy/c.asp?f=css_box-shadow&p=15) |

5.transition

transition: *property* *duration* *timing-function* *delay*;

|  |  |
| --- | --- |
| 值 | 描述 |
| [transition-property](http://www.w3school.com.cn/cssref/pr_transition-property.asp) | 规定设置过渡效果的 CSS 属性的名称。 |
| [transition-duration](http://www.w3school.com.cn/cssref/pr_transition-duration.asp) | 规定完成过渡效果需要多少秒或毫秒。 |
| [transition-timing-function](http://www.w3school.com.cn/cssref/pr_transition-timing-function.asp) | 规定速度效果的速度曲线。 |
| [transition-delay](http://www.w3school.com.cn/cssref/pr_transition-delay.asp) | 定义过渡效果何时开始。 |

34.display：box-flex弹性布局

box-flex：1；分得1份 均分剩余宽度

box-pack水平对齐方式：start左对齐 | center居中对齐 | end右对齐 | justify分散布局

box-align垂直对齐方式：start（top） | center（middle） | end（bottom） | baseline基线 | stretch（默认值）

box-direction：normal | reverse是否反转

box-orient排列方式：horizontal水平 | vertical垂直 改变pack和align方向

**display：flex弹性布局**

**flex**：1均分宽度

flex-grow扩展比率

剩余空间是正值的时，伸缩项目相对于伸缩容器里其他伸缩项目能分配到空间比例,若没写该属性，则为0,0代表不参与扩展

flex-shrink缩小比例

此时剩余空间是不足时都将等比例缩小,0 表示不参加收缩比例,若没写该属性，则为1

flex-basis

在分配空间之前，也已经分得到空间

（最小宽度）

缩写 flex: none 或者 [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]（||后面的属性可不写）

**flex-direction**排列方式，决定主轴位置：row | row-reverse | **column** | column-reverse

水平，从左起 | 水平从右起 | 垂直从上起 | 垂直从下起

flex-wrap: nowrap | wrap | wrap-reverse

单行 | 换行(宽度不够自动换行) | 换行并反转

**justify-content**（水平对齐方式）

flex-start | flex-end | center | space-between | space-around;

左对齐 | 右对齐 | 居中对齐 | 两端对齐(之间的间隔相等) | 每个项目两侧的间隔相等

**align-items**（垂直对齐方式）

flex-start | flex-end | center | baseline | stretch

起点对齐 | 终点对齐 | 中点对齐 | 基线对齐 | 默认值(未设置高度或设为auto，将占满整个容器的高度)

align-content （多行对齐方式）

注：需要多行才行

多根轴线的对齐方式

flex-start | flex-end | center | space-between | space-around | stretch

左对齐 | 右对齐 | 居中对齐 | 两端对齐(之间的间隔相等) | 每个项目两侧的间隔相等 | 轴线占满整个交叉轴

align-self

允许单个项目有与其他项目不一样的对齐方式

auto | flex-start | flex-end | center | baseline | stretch

该属性可能取6个值，除了auto，其他都与align-items属性完全一致。

可覆盖align-items属性

默认值auto，表示继承父级元素的align-items，如果没有父级，则等同于stretch

当这两个属性遇到了

flex-direction: column / column-reverse 整个顺序就都变了。

36. white-space 属性设置如何处理元素内的空白。

|  |  |
| --- | --- |
| **值** | **描述** |
| normal | 默认。空白会被浏览器忽略。 |
| pre | 空白会被浏览器保留。其行为方式类似 HTML 中的 <pre> 标签。 |
| nowrap | 文本不会换行，文本会在在同一行上继续，直到遇到 <br> 标签为止。 |
| pre-wrap | 保留空白符序列，但是正常地进行换行。 |
| pre-line | 合并空白符序列，但是保留换行符。 |
| inherit | 规定应该从父元素继承 white-space 属性的值。 |

**-webkit-user-modify:**

read-only（只读） | read-write（可读写） | read-write-plaintext-only（内容可读写，但粘贴内容中的富文本格式（如文本的颜色、大小，图片等）会丢失。内容类似于以纯文本显示。）

标签属性contenteditable=”true”可读写

**-webkit-user-select:**

auto（用户可选内容） | none（不可选） | text（可选文本）

**-webkit-box-relect：（倒影）**

direction：above上|below下|left左|right右

offset：number|%

mask-box-image（设置倒影的遮罩图像）

url | linear-gradient | radial-gradient | repeating-linear-gradient | repeating-url

38.

**-webkit-filter**

-webkit-filter: blur(3px);模糊效果

-webkit-filter: brightness(.5);亮度

filter: contrast(.5);对比度

filter: grayscale(.5);灰度

filter: hue-rotate(0deg);色相旋转

filter: saturate(.5);反色

filter: sepia(.5);透明度

filter: drop-shadow(x y radius color);褐色程度

filter: drop-shadow(30px 30px 20px red);阴影（有圆角）

box-shadow: 30px 30px 20px red;阴影（无圆角）

39.animation

animation-name 名称

animation-duration 执行时间

animation-timing-function 运动类型(ease-in等)

animation-delay 延迟

animation-iteration-count 循环次数 可选 number | infinite

animation-direction 是否反向运动 alternate

animation-fill-mode

设置或检索动画时间之外的状态（想让动画保持突然终止时的状态）

none | forwards | backwards | both

不设置 | 结束时 | 开始时 | 结束或开始的状态

animation-play-state 想让动画保持突然终止时的动画状态 running | paused

@keyframes name{

0% { background: #c00 }

50% { background: orange }

100% { background: yellowgreen }

}

42.table表格

<table border="1" width="600" frame="hsides" rules="groups">  
     <caption>My Ultimate Table</caption>  
     <colgroup span="1" width="200"></colgroup>  
     <colgroup span="3" width="400"></colgroup>  
<thead>  
     <tr>  
          <td>cell 1-1</td>  
          <td>cell 1-2</td>  
          <td>cell 1-3</td>  
          <td>cell 1-4</td>  
     </tr>  
</thead>  
<tfoot>  
     <tr>  
          <td>cell 4-1</td>  
          <td>cell 4-2</td>  
          <td>cell 4-3</td>  
          <td>cell 4-4</td>  
     </tr>  
</tfoot>  
<tbody>  
     <tr>  
          <td>cell 2-1</td>  
          <td>cell 2-2</td>  
          <td>cell 2-3</td>  
          <td>cell 2-4</td>  
     </tr>  
     <tr>  
          <td>cell 3-1</td>  
          <td>cell 3-2</td>  
          <td>cell 3-3</td>  
          <td>cell 3-4</td>  
     </tr>  
</tbody>  
</table>

1）thead和tfoot在一张表中都只能有一个，而tbody可以有多个。

2）tfoot必须出现在tbody前面，这样浏览器在接收主体数据之前，就能渲染表尾，有利于加快表格的显示速度。这一点对大型表格尤其重要。

3）thead、tbody和tfoot里面都必须使用tr标签。

表示单元格的标签是th（table head）和td（table data），前者用来显示数据的名称，后者用来显示数据的内容。

tbody可以用来对"行"进行分组，colgroup则用来对"列"进行分组，span属性，可以指定colgroup标签能够影响到的列数。

frame属性可取的值及含义如下：

\* void - 默认值。表示不显示表格最外围的边框。  
\* box - 同时显示四条边框。  
\* border - 同时显示四条边框。  
\* above - 只显示顶部边框。  
\* below - 只显示底部边框。  
\* lhs - 只显示左侧边框。  
\* rhs - 只显示右侧边框。  
\* hsides - 只显示水平方向的两条边框。  
\* vsides - 只显示垂直方面的两条边框。

rules 属性可取的值有五个，分别是：

\* none - 默认值。无边框。  
\* groups - 为行组或列组加边框。  
\* rows - 为行加边框。  
\* cols - 为列加边框。  
\* all - 为所有行列（单元格）加边框

43.自适应div垂直居中的方式

<style type="text/css">

div{width:200px;height:200px;border:1px solid #000;text-align:center;

vertical-align:middle;display:table-cell;}

span{vertical-align:middle;}

</style>

</head>

<body>

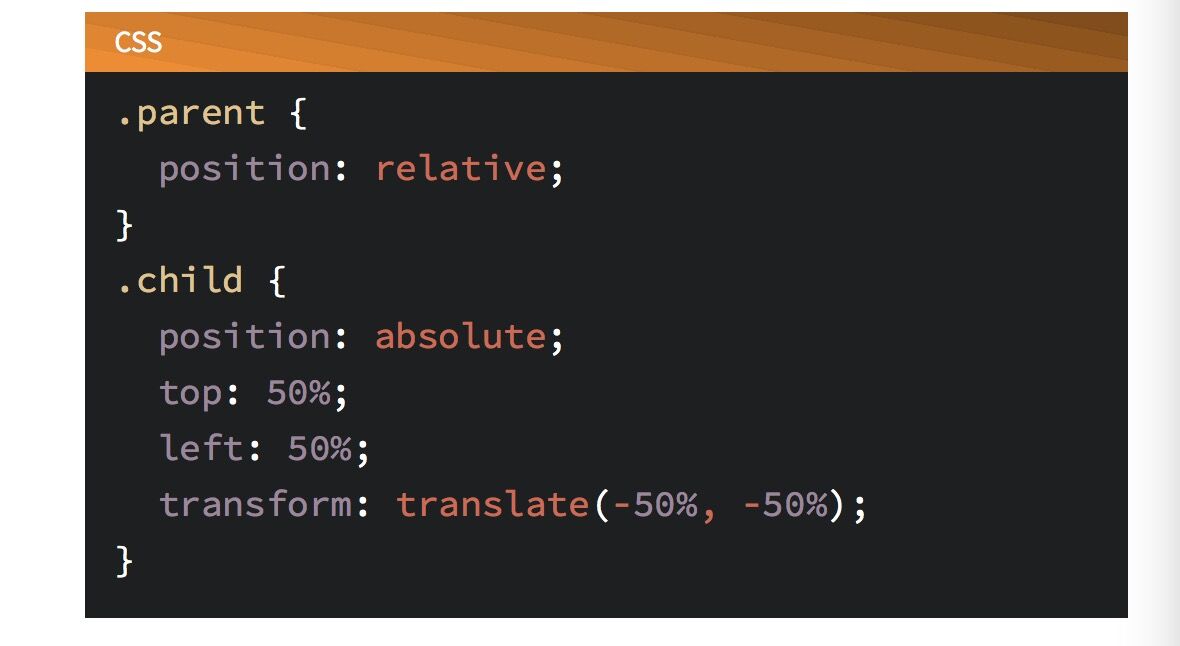
<div>

<span>17:34:55</span>

</div>

</body>

# position & transform: translate(-50%, -50%) 实现块元素百分比下居中



利用flexbox

.parent{

　　　　justify-content:center;

　　　　align-items:center;

　　　　display: flex;

　　}

利用定位与margin：auto

.parent{

width: 600px;

height: 400px;

background: red;

position: relative;

}

.children{

width: 200px;

height: 200px;

position: absolute;

top: 0;

left: 0;

bottom: 0;

right: 0;

margin: auto;

background: black;

}

<style type="text/css">

div{width:200px;height:200px;border:1px solid #000;text-align:center;}

i{display:inline-block;height:100%;vertical-align:middle;}

span{vertical-align:middle;}

</style>

</head>

<body>

<div>

<i></i>

<span>17:34:55</span>

</div>

</body>

45.伪类

a:link {color:black;}

a:visited {color:gray;}

a:hover {text-decoration:none;}

a:active {color:red;}

input:focus {border:1px solid blue;}

如果用户点击一个指向页面中其他元素的链接，则那个元素就是目标（target），可以用 :target 伪类选中它。（<a href="#more\_info">More Information</a>）

#more\_info:target {background:#eee;}

:first-child 和 :last-child

:nth-child

一个冒号（ : ）表示伪类，两个冒号（ :: ）表示 CSS3新增的伪元素。

伪元素

::first-letter 伪元素

首字符下沉

p::first-letter {

font-family:Lato, helvetica, sans-serif;

font-style: normal;

font-weight:700;

font-size:4.5em;

float:left;

line-height:0.5;

}

::first-line 伪元素

::before 和 ::after 伪元素

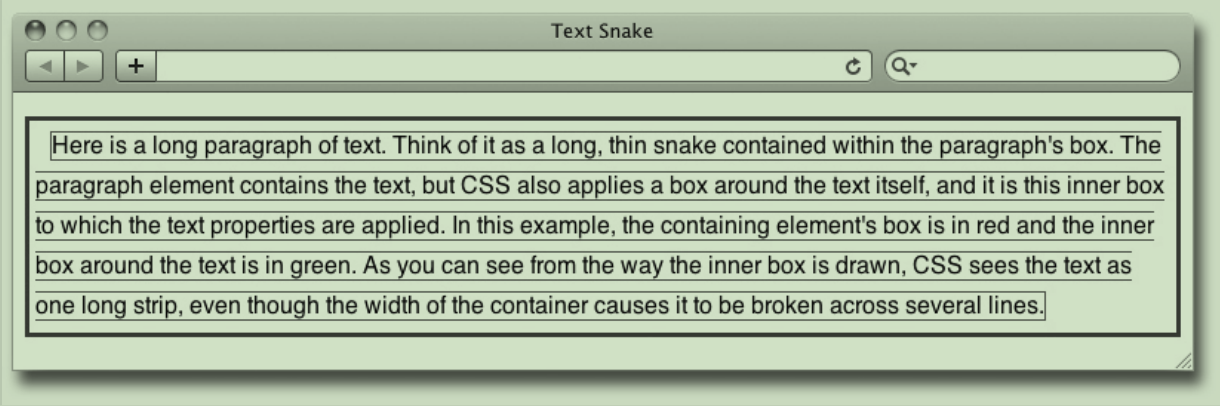
字体

font-style：italic 、 oblique 、 normal

font-weight：100 、 200 …… 900 ，或者 lighter 、 normal 、 bold 和 bolder

font-variant：small-caps将所有小写英文字母变成小型大写字母

文本

文本属性只应用于这个长长的、细细的、内部的文本盒子，而不是包含元素的盒子。

 text-indent文本缩进

 letter-spacing字符间距

 word-spacing单词间距

 text-decoration文本装饰（underline 、 overline 、 line-through 、 blink 、 none ）

 text-align

 line-height

 text-transform文本转换（none 、 uppercase 、 lowercase 、 capitalize用于转换元素中文本的大小写，它可以设定英文单词首字母大

写、全部字母大写和全部字母小写。）

 vertical-align以基线为参照上下移动文本，但这个属性只影响行内元素。如果你想在垂直方向上对齐块级元素，必须把其 display 属性设定为 inline 。 vertical-align属性最常用于公式或化学分子式中的上标和下标，比如 x 4 -y -5 或 N 3 O，或者用于文本中脚注的角标，比如把星号变成上角标。浏览器默认<sup>上标，<sub>为下标

三栏布局

固定宽度布局：三个div设置宽度，然后float，边距可以用内嵌一个div或者是设置box-sizing：border-box来防止布局混乱（直接设置边距会导致宽度增加，浮动会下移）

流体布局：

<div class**=**"container">

<div class**=**"left"></div>

<div class**=**"right"></div>

<div class**=**"main"></div>

</div>

.left和.right固定宽度并左右浮动，.main不定宽度，设置外边距

BFC三栏布局：

<div class**=**"container">

<div class**=**"left"></div>

<div class**=**"right"></div>

<div class**=**"main"></div>

</div>

.left和.right固定宽度并左右浮动，.main不定宽度，设置overflow，成BFC

双飞翼布局

**.content** {

float**:** left;

width**:** 100%;

}

**.main** {

height**:** 200px;

margin-left**:** 110px;

margin-right**:** 220px;

background-color**:** green;

}

**.left** {

float**:** left;

height**:** 200px;

width**:** 100px;

margin-left**:** -100%;

background-color**:** red;

}

**.right** {

width**:** 200px;

height**:** 200px;

float**:** right;

margin-left**:** -200px;

background-color**:** blue;

}

<div class**=**"content">

<div class**=**"main"></div>

</div>

<div class**=**"left"></div>

<div class**=**"right"></div>

圣杯布局

**.container** {

margin-left**:** 120px;

margin-right**:** 220px;

}

**.main** {

float**:** left;

width**:** 100%;

height**:** 300px;

background-color**:** red;

}

**.left** {

float**:** left;

width**:** 100px;

height**:** 300px;

margin-left**:** -100%;

position**:** relative;

left**:** -120px;

background-color**:** blue;

}

**.right** {

float**:** left;

width**:** 200px;

height**:** 300px;

margin-left**:** -200px;

position**:** relative;

right**:** -220px;

background-color**:** green;

}

<div class**=**"container">

<div class**=**"main"></div>

<div class**=**"left"></div>

<div class**=**"right"></div>

</div>

Flex布局

<div class**=**"container">

<div class**=**"main"></div>

<div class**=**"left"></div>

<div class**=**"right"></div>

</div>

.left和.right固定宽度，.left设置order：-1排在前面，.main设置flex：1

Table布局

<div class**=**"container">

<div class**=**"left"></div>

<div class**=**"main"></div>

<div class**=**"right"></div>

</div>

.left.right.main设置display：table-cell，.left.right固定宽度，缺点：无法设置间距

绝对定位布局

<div class**=**"container">

<div class**=**"main"></div>

<div class**=**"left"></div>

<div class**=**"right"></div>

</div>

.main用margin：0 x居中，x为留给.left.right的宽度，.left.right绝对定位到左上，右上（left：0，top：0，right：0）

ul li + li{border-top:1px solid #000;}

第一个li不加border-top，其余都加上

String包装类型

1、字符串转换

字符串转换是最基础的要求和工作，你可以将任何类型的数据都转换为字符串，你可以用下面三种方法的任何一种：

var num= 19; // 19

var myStr = num.toString(); // "19"

你同样可以这么做：

var num= 19; // 19

var myStr = String(num); // "19"

或者，再简单点儿：

var num= 19; // 19

var myStr = "" +num; // "19"

2、字符串分割

字符串分割，即将一个字符串分割为多个字符串，JavaScript中给我们提供了一个非常方便的函数，如：

var myStr = "I,Love,You,Do,you,love,me";

var substrArray = myStr .split(","); // ["I", "Love", "You", "Do", "you", "love", "me"];

var arrayLimited = myStr .split(",", 3); // ["I", "Love", "You"];

split()的第二个参数，表示返回的字符串数组的最大长度。

3、获取字符串长度

字符串长度是在开发中经常要用到的，非常简单如下：

var myStr = "I,Love,You,Do,you,love,me";

var myStrLength = myStr.length; //25

4、查询子字符串

很多人都会忘记这几个JavaScript的自带的方法，或者忘记它们的具体用法，从而导致在做题的时候不得不嵌套for循环来做。

第一个函数：indexOf()，它从字符串的开头开始查找，找到返回对应坐标，找不到返回-1。如下：

var myStr = "I,Love,you,Do,you,love,me";

var index = myStr.indexOf("you"); // 7 ,基于0开始,找不到返回-1

第二个函数：lastIndexOf()，它从字符串的末尾开始查找，找到返回对应坐标，找不到返回-1。如下：

var myStr = "I,Love,you,Do,you,love,me";

var index = myStr.lastIndexOf("you"); // 14

以上两个函数同样接收第二个可选的参数，表示开始查找的位置。

5、字符串替换

单单查到字符串应该还不会停止，一般题目都还经常会遇到让你查到并替换为你自己的字符串，例如：

var myStr = "I,love,you,Do,you,love,me";

var replacedStr = myStr.replace("love","hate");//"I,hate,you,Do,you,love,me"

默认只替换第一次查找到的，想要全局替换，需要置上正则全局标识，如：

var myStr = "I,love,you,Do,you,love,me";

var replacedStr = myStr.replace(/love/g,"hate");//"I,hate,you,Do,you,hate,me"

6、查找给定位置的字符或其字符编码值

想要查找给定位置的字符，你可以使用如下函数：

var myStr = "I,love,you,Do,you,love,me";

var theChar = myStr.charAt(8);// "o",同样从0开始

同样，它的一个兄弟函数就是查找对应位置的字符编码值，如：

var myStr = "I,love,you,Do,you,love,me";

var theChar = myStr.charCodeAt(8); //111

7、字符串连接

字符串连接操作可以简单到用一个加法运算符搞定，如：

var str1 = "I,love,you!";

var str2 = "Do,you,love,me?";

var str = str1 + str2 + "Yes!";//"I,love,you!Do,you,love,me?Yes!"

同样，JavaScript也自带了相关的函数，如：

var str1 = "I,love,you!";

var str2 = "Do,you,love,me?";

var str = str1.concat(str2);//"I,love,you!Do,you,love,me?"

其中concat()函数可以有多个参数，传递多个字符串，拼接多个字符串。

8、字符串切割和提取

有三种可以从字符串中抽取和切割的方法，如：

第一种，使用slice():

var myStr = "I,love,you,Do,you,love,me";

var subStr = myStr.slice(1,5);//",lov"

第二种，使用substring():

var myStr = "I,love,you,Do,you,love,me";

var subStr = myStr.substring(1,5); //",lov"

第三种，使用substr():

var myStr = "I,love,you,Do,you,love,me";

var subStr = myStr.substr(1,5); //",love"

与第一种和第二种不同的是，substr()第二个参数代表截取的字符串最大长度，如上结果所示。

9、字符串大小写转换

常用的转换为大写或者小写字符串函数，如下：

var myStr = "I,love,you,Do,you,love,me";

var lowCaseStr = myStr.toLowerCase();//"i,love,you,do,you,love,me";

var upCaseStr = myStr.toUpperCase();//"I,LOVE,YOU,DO,YOU,LOVE,ME"

10、字符串匹配

字符串匹配可能需要你对正则表达式有一定的了解，先来看看match()函数：

var myStr = "I,love,you,Do,you,love,me";

var pattern = /love/;

var result = myStr.match(pattern);//["love"]

console.log(result .index);//2

console.log(result.input );//I,love,you,Do,you,love,me

如你所见，match()函数在字符串上调用，并且接受一个正则的参数。来看看第二个例子，使用exec()函数：

var myStr = "I,love,you,Do,you,love,me";

var pattern = /love/;

var result = pattern .exec(myStr);//["love"]

console.log(result .index);//2

console.log(result.input );//I,love,you,Do,you,love,me

简单吧，仅仅是把正则和字符串换了个位置，即exec()函数是在正则上调用，传递字符串的参数。对于上面两个方法，匹配的结果都是返回第一个匹配成功的字符串，如果匹配失败则返回null.

再来看一个类似的方法search(),如：

var myStr = "I,love,you,Do,you,love,me";

var pattern = /love/;

var result = myStr.search(pattern);//2

仅返回查到的匹配的下标，如果匹配失败则返回-1.

11、字符串比较

比较两个字符串，比较是规则是按照字母表顺序比较的，如：

var myStr = "chicken";

var myStrTwo = "egg";

var first = myStr.localeCompare(myStrTwo); // -1

first = myStr.localeCompare("chicken"); // 0

first = myStr.localeCompare("apple"); // 1

12、举例

最后我们来看一道前端笔试题，去哪儿网的，相信很多孩子都做到过这个题了。题目：写一个getSuffix函数，用于获得输入参数的后缀名，例如输入abcd.txt，返回txt。附上我的答案：

function getSuffix(file){

return file.slice(file.lastIndexOf(".") + 1,file.length);

}

concat（）连接字符

trim（）删除字符开头和结尾的空格，返回字符副本

获取类

charAt(index)指定字符

charCodeAt指定字符编码、fromCharCode执行相反操作

查找类

indexOf()、lastIndexOf()、search()正则--查找

match、replace—匹配和替换

截取类

slice(1,3)、substr(1,3)、substring(1,3)

substr的第二个参数是返回字符的个数，slice、substring则是index(0开始)

3个都返回基本数据类型的字符串值，对原始值无影响

其他

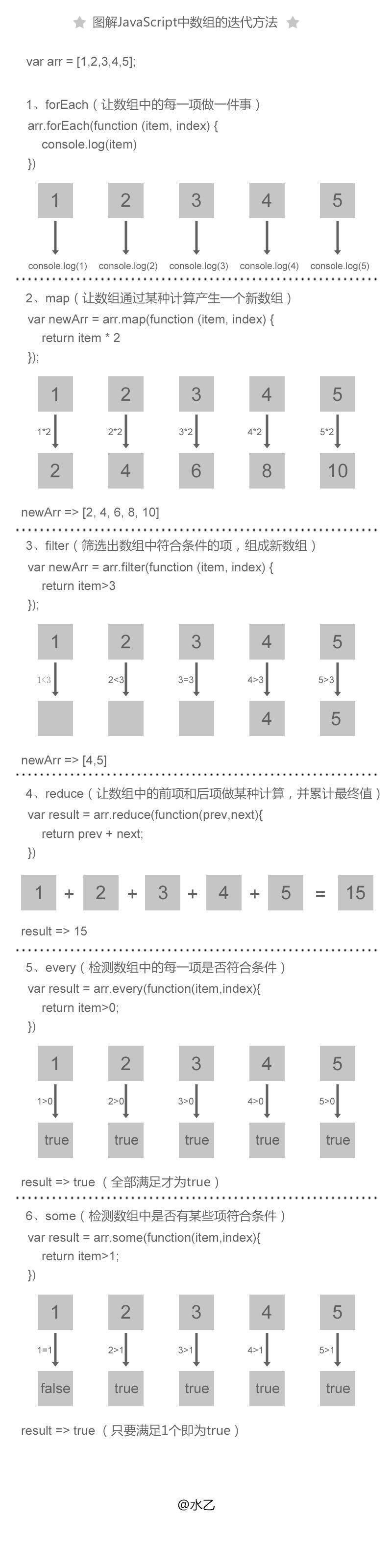
split—根据分隔符，拆分数组

toLowerCase、toUpperCase

match（）、search（）、replace（）用于在字符串中匹配模式的方法

localeCompare（参数）比较字符串大小，参数排在前返回1

21.array



从尾部添加push

从尾部删除pop

从头部删除shift

从头部添加unshift

转换、获取

splice（开始位置，长度，元素） 先删除一段元素，再添加一段元素

join 与split用法相反

slice

排序

数字

sort(function(num1,num2){

return num1-num2;

})

汉字

sort(function(str1,str2){

return str1.localeCompare(str2);})

reverse反转

其他

arr1.concat(arr2)数组连接

+连接后的的类型是string

es5数组方法：通过 . 来调用

forEach（数组元素，元素索引，数组）

map（每个数组元素），返回新数组，不修改调用的函数

reduce() 方法接收一个函数作为累加器，数组中的每个值（从左到右）开始缩减，最终计算为一个值。

filter（），跳过稀疏数组中缺少的元素，返回数组总是稠密的。新数组中的元素是通过检查指定数组中符合条件的所有元素。返回布尔值

every（），当且仅当针对数组中的所有元素调用判定函数都返回true，才返回true

some（），当数组中至少有一个元素调用判定函数返回true，就返回true

indexOf（）、lastIndexOf（）搜索整个数组中具有给定值的元素，返回找到的第一个元素的索引，若无则返回-1

判定类型Array.isArray（）

Object.prototype.toString.call(obj)

console.log(Object.prototype.toString.call(123)) //[object Number]  
console.log(Object.prototype.toString.call('123')) //[object String]  
console.log(Object.prototype.toString.call(undefined)) //[object Undefined]  
console.log(Object.prototype.toString.call(true)) //[object Boolean]  
console.log(Object.prototype.toString.call({})) //[object Object]  
console.log(Object.prototype.toString.call([])) //[object Array]  
console.log(Object.prototype.toString.call(function(){})) //[object Function]

typeof检测基本类型：**number**、**string**、**boolean**、**null**（“object“）、**undefined**、function（”function“）

**instanceof** **运算符**用来测试一个对象在其原型链中是否存在一个构造函数的 prototype 属性。

引用类型：Object、Array、Date、RegExp、Function、基本包装类型（Boolean、Number、String）、单体内置对象（Global、Math）

2+.DOM操作

一、DOM创建

DOM节点（Node）通常对应于一个标签，一个文本，或者一个HTML属性。DOM节点有一个nodeType属性用来表示当前元素的类型，它是一个整数：

Element，元素

Attribute，属性

Text，文本

DOM节点创建最常用的便是document.createElement和document.createTextNode方法：

var node1 = document.createElement('div');

var node2 = document.createTextNode('hello world!');

二、DOM查询

// 返回当前文档中第一个类名为 "myclass" 的元素

var el = document.querySelector(".myclass");

// 返回一个文档中所有的class为"note"或者 "alert"的div元素

var els = document.querySelectorAll("div.note, div.alert");

// 获取元素

var el = document.getElementById('xxx');

var els = document.getElementsByClassName('highlight');

var els = document.getElementsByTagName('td');

Element也提供了很多相对于元素的DOM导航方法：

// 获取父元素、父节点

var parent = ele.parentElement;

var parent = ele.parentNode;//只读，没有兼容性问题

var offsetParent=ele.offsetParent;//只读，找到最近的有定位的父节点。

　　　　　　　　　　　　　　　　　　　　　//没有定位父级时，默认是body;但在IE7以下，如果当前元素没有定位属性，返回body，如果有，返回HTML;

　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　 //如果当前元素某个父级触发了haslayout，则返回触发了haslayout这个元素。

// 获取子节点，子节点可以是任何一种节点，可以通过nodeType来判断

var nodes = ele.children;//标准下、非标准下都只含元素类型，但对待非法嵌套的子节点，处理方式与childNodes一致。

var nodes = ele.childNodes;//非标准下：只包含元素类型，不会包含非法嵌套的子节点。

　　　　　　　　　　　　　　　　//标准下：包含元素和文本类型，会包含非法嵌套的子节点。

//获取元素属性列表

var attr = ele.attributes;

// 查询子元素

var els = ele.getElementsByTagName('td');

var els = ele.getElementsByClassName('highlight');

// 当前元素的第一个/最后一个子元素节点

var el = ele.firstChild;//对待标准和非标准模式，如childNods

var el = ele.lastChild;

var el = ele.firstElementChild;//非标准不支持

var el = ele.lastElementChild;

// 下一个/上一个兄弟元素节点

var el = ele.nextSibling;

var el = ele.previousSibling;

var el = ele.nextElementSibling;

var el = ele.previousElementSibling;

兼容的获取第一个子元素节点方法：

var first=ele.firstElementChild||ele.children[0];

三、DOM更改

// 添加、删除子元素

ele.appendChild(el);

ele.removeChild(el);

// 替换子元素

ele.replaceChild(el1, el2);

// 插入子元素

parentElement.insertBefore(newElement, referenceElement);

//克隆元素

ele.cloneNode(true) //该参数指示被复制的节点是否包括原节点的所有属性和子节点

四、属性操作

// 获取一个{name, value}的数组

var attrs = el.attributes;

// 获取、设置属性

var c = el.getAttribute('class');

el.setAttribute('class', 'highlight');

// 判断、移除属性

el.hasAttribute('class');

el.removeAttribute('class');

// 是否有属性设置

el.hasAttributes();

innerHTML与outerHTML的区别？

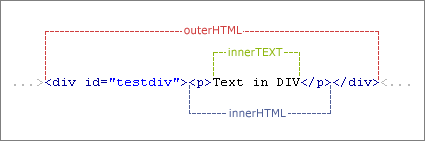
比如对于这样一个HTML元素：<div>content<br/></div>。

innerHTML：内部HTML，content<br/>；

outerHTML：外部HTML，<div>content<br/></div>；

innerText：内部文本，content；

outerText：内部文本，content；

上述四个属性不仅可以读取，还可以赋值。outerText和innerText的区别在于outerText赋值时会把标签一起赋值掉，另外xxText赋值时HTML特殊字符会被转义。

jQuery的html()与innerHTML的区别？

jQuery的.html()会调用.innerHTML来操作，但同时也会catch异常，然后用.empty(), .append()来重新操作。 这是因为IE8中有些元素(比如input等)的.innerHTML是只读的。

创建新节点：createDocumentFragment、createElement、createTextNode

添加、移除、替换、插入：

appendChild（1.先移出父级 2.添加到新的父级）、removeChild、replaceChild、insertBefore

复制节点：cloneNode（true），true为深复制

获取、设置（修改）、删除属性：getAttribute、setAttribute、removeAttribute

获取子节点

firstChild、lastChild、childNodes[0]（childNodes.item(0)）会获取空白节点，不推荐

firstElementChild、lastElementChild、children[0]取代上述属性

新增API，可以通过Document和Element类型的实例调用

querySelector()、querySelectorAll()接收一个CSS选择符参数

querySelectorAll()返回一个NodeList对象，可以用[]和item()取得元素

27.正则表达式

search查找 substring获取子字符串 charAt获取某个字符 split分割字符

match 获取匹配的字符 replace替换字符

js风格 new RegExp(”a”,”i”)

perl风格 /a/i

选项

i=ignore g=global全部的

量词

{n,m}最少n次，最多m次

{n, }最少n次，最多不限

{ ,m}最少不限，最多m次

{n}正好n次

+等于｛1，｝

\*等于｛0，｝

？等于｛0,1｝

字符类

任意字符：[abc] 例子 1[abc]2----1a2 1b2 1c2

范围：[0-9]、[a-z] 例子 id[0-9]----id0、id7

排除：[^a] 除了a以外所有

转义字符：

. 任意字符（包括所有字符）

\d=digital数字 \w [a-z0-9\_] \s space空格

\D=digital除了数字 \W 除了[a-z0-9\_] \S 除了space空格

^行首 $行尾

[\u4e00]-[\u9fa5]汉字

28.offset scroll client

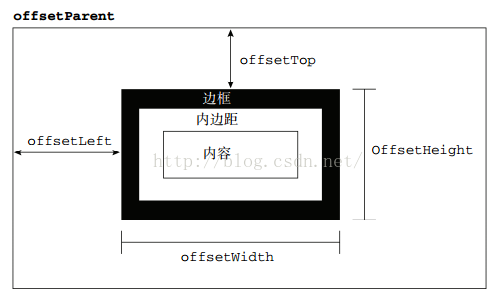
offsetWidth和offsetHeight用来获得对象的大小，由自身宽高、内边距、边框组成。

**offsetHeight和style.height的区别**

1> style.height只能获取行内样式**，**offsetHeight可以获取行内样式和内嵌样式

2> style.height是字符串（而且带单位），offsetHeight是数值

3> style.height可读可写，offsetHeight是只读属性



offsetLeft和offsetTop用来获得对象的位置，到距离自身最近的（带有定位的）父元素的距离，若没有则以body为准。

**offsetLeft和style.left的区别**

1> style.left只能获取行内样式，offsetLeft 可以获取行内样式和内嵌样式

2> offsetLeft 只读，style.left可读可写

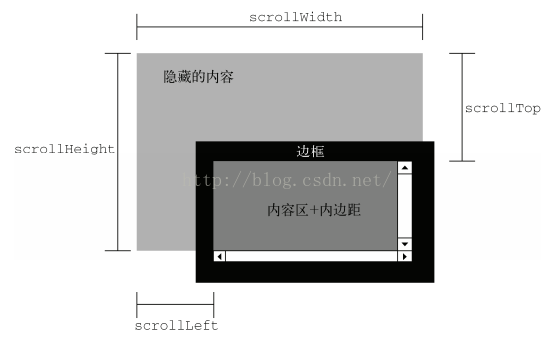
3> offsetLeft是数值，style.left是字符串并且有单位px

4> 如果没有加定位，style.left获取的数值可能是无效的

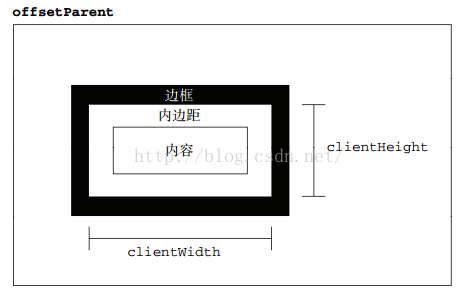
5> 最大区别在于offsetLeft以border左上角为基准，style.left以margin左上角为基准

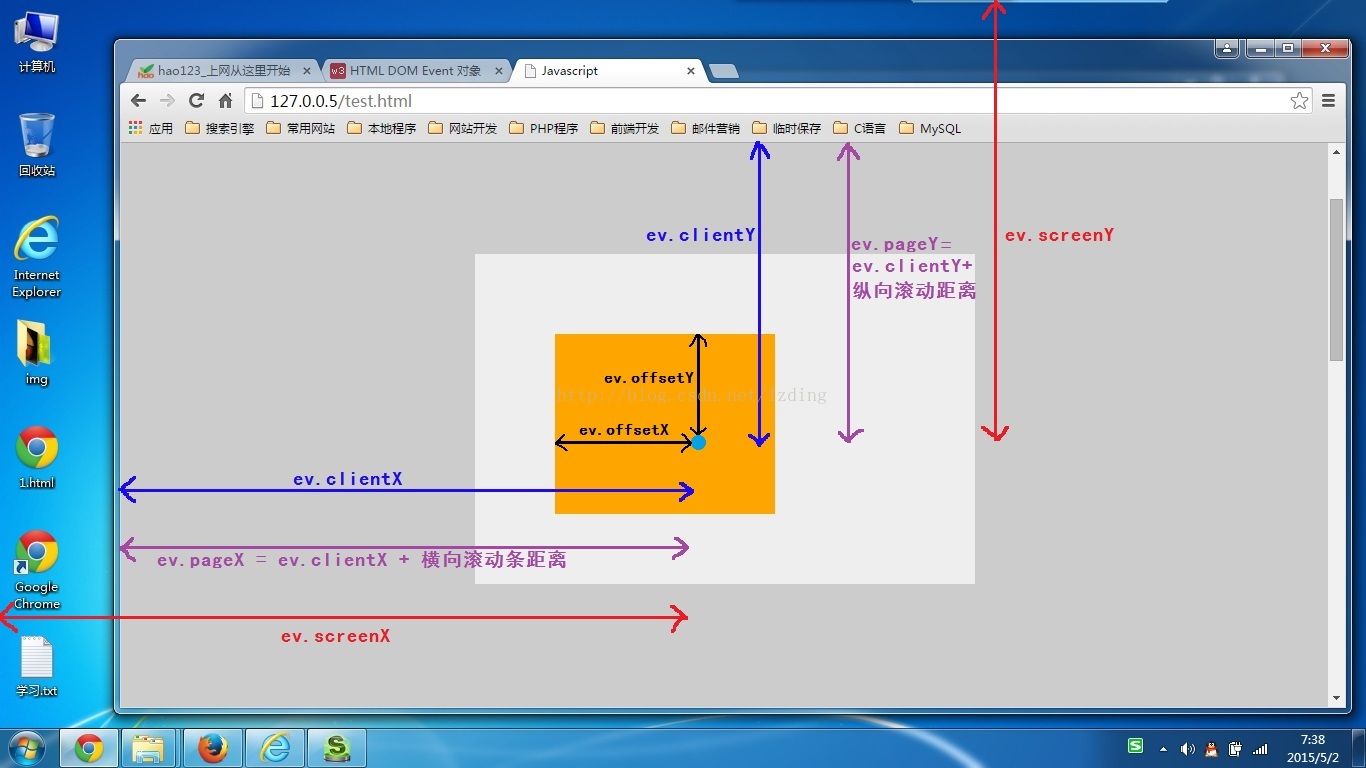
scrollHeight和scrollWidth对象内部实际内容的高度、宽度，包括内容和内边距，不包括边框。

scrollLeft和scrollTop显示内容区到整个内容区的距离



clientWidth和clientHeight包括内容区和内边距，不包括边框





30.同步与异步

Js只有一条主线程

**同步任务指的是，在主线程上排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务；**

**异步任务指的是，不进入主线程、而进入"任务队列"（task queue）的任务，只有等主线程任务执行完毕，"任务队列"开始通知主线程，请求执行任务，该任务才会进入主线程执行。**

**具体来说，异步运行机制如下：**

**（1）所有同步任务都在主线程上执行，形成一个执行栈（execution context stack）。  
（2）主线程之外，还存在一个"任务队列"（task queue）。只要异步任务有了运行结果，就在"任务队列"之中放置一个事件。  
（3）一旦"执行栈"中的所有同步任务执行完毕，系统就会读取"任务队列"，看看里面有哪些事件。那些对应的异步任务，于是结束等待状态，进入执行栈，开始执行。  
（4）主线程不断重复上面的第三步。**

31.检测屏幕旋转

window.orientation===’-90’||window.orientation===’90’

32.

a["城市"]="北京";

a["面积"]=16800;

a["人口"]=1600;

在Javascript语言中，关联数组就是对象，对象就是关联数组。这一点与php语言完全不同，在php中，关联数组也是数组。

33.闭包中的this指向window

因为this指向调用函数的对象，闭包是函数内的函数，严格模式下为undefind

34.如果某个microtask任务被推入到执行中，那么当主线程任务执行完成后，会循环调用该队列任务中的下一个任务来执行，直到该任务队列到最后一个任务为止。而事件循环每次只会入栈一个macrotask,主线程执行完成该任务后又会检查microtasks队列并完成里面的所有任务后再执行macrotask的任务。

整体的js代码这个macrotask先执行，同步代码执行完后有microtask执行microtask，没有microtask执行下一个macrotask，如此往复循环至结束

microtasks: process.nextTick, Promises, Object.observe, MutationObserver

macrotasks: script(整体代码),setTimeout, setInterval, setImmediate, I/O, UI rendering

1.降低请求量：合并资源，减少 HTTP 请求数，minify / gzip 压缩，webP，lazyLoad。  
2.加快请求速度：预解析DNS，减少域名数，并行加载（多个域名下内容），CDN 分发。  
3.缓存：HTTP 协议缓存请求，离线缓存 manifest，离线数据缓存localStorage。  
4.渲染：JS/CSS优化，加载顺序，服务端渲染，pipeline。  
  
两种缓存：  
1.询问是否有更新：根据 If-Modified-Since / ETag 等协议向后端请求询问是否有更新，没有更新返回304，浏览器使用本地缓存。  
2.直接使用本地缓存：根据协议里的 Cache-Control / Expires 字段去确定多长时间内可以不去发请求询问更新，直接使用本地缓存。

分域名请求图片的原因和好处

浏览器的并发请求数目限制是针对同一域名的，超过限制数目的请求会被阻塞

浏览器并发请求有个数限制，分域名可以同时并发请求大量图片

HTTP头信息控制缓存

Expires（强缓存）+过期时间   Expires是HTTP1.0提出的一个表示资源过期时间的header，它描述的是一个绝对时间

Cache-control（强缓存） 描述的是一个相对时间，在进行缓存命中的时候，都是利用客户端时间进行判断 管理更有效，安全一些 Cache-Control: max-age=3600

Last-Modified/If-Modified-Since（协商缓存） 标示这个响应资源的最后修改时间。Last-Modified是服务器相应给客户端的，If-Modified-Sinces是客户端发给服务器，服务器判断这个缓存时间是否是最新的，是的话拿缓存。

Etag/If-None-Match（协商缓存） etag和last-modified类似，他是发送一个字符串来标识版本。

// 函数节流

var canRun = true;

document.getElementById("throttle").onscroll = function(){

if(!canRun){

// 判断是否已空闲，如果在执行中，则直接return

return;

}

canRun = false;

setTimeout(function(){

console.log("函数节流");

canRun = true;

}, 300);

};

// 函数防抖

var timer = false;

document.getElementById("debounce").onscroll = function(){

clearTimeout(timer); // 清除未执行的代码，重置回初始化状态

timer = setTimeout(function(){

console.log("函数防抖");

}, 300);

};

预加载：

function loadImage(url, callback) {

var img = new Image();

img.src = url;

if (img.complete) { // 如果图片已经存在于浏览器缓存，直接调用回调函数 防止IE6不执行onload BUG

callback.call(img);

return;

}

img.onload = function () {

callback.call(img);//将回调函数的this替换为Image对象

};

};

懒加载：

当网页滚动的事件被触发 -> 执行加载图片操作 -> 判断图片是否在可视区域内 -> 在，则动态将data-src的值赋予该图片。

var aImages = document.getElementById("SB").getElementsByTagName('img'); //获取id为SB的文档内所有的图片

loadImg(aImages);

window.onscroll = function() { //滚动条滚动触发

loadImg(aImages);

};

//getBoundingClientRect 是图片懒加载的核心

function loadImg(arr) {

for(var i = 0, len = arr.length; i < len; i++) {

if(arr[i].getBoundingClientRect().top < document.documentElement.clientHeight && !arr[i].isLoad) {

arr[i].isLoad = true; //图片显示标志位

//arr[i].style.cssText = "opacity: 0;";

(function(i) {

setTimeout(function() {

if(arr[i].dataset) { //兼容不支持data的浏览器

aftLoadImg(arr[i], arr[i].dataset.imgurl);

} else {

aftLoadImg(arr[i], arr[i].getAttribute("data-imgurl"));

}

arr[i].style.cssText = "transition: 1s; opacity: 1;" //相当于fadein

}, 500)

})(i);

}

}

}

function aftLoadImg(obj, url) {

var oImg = new Image();

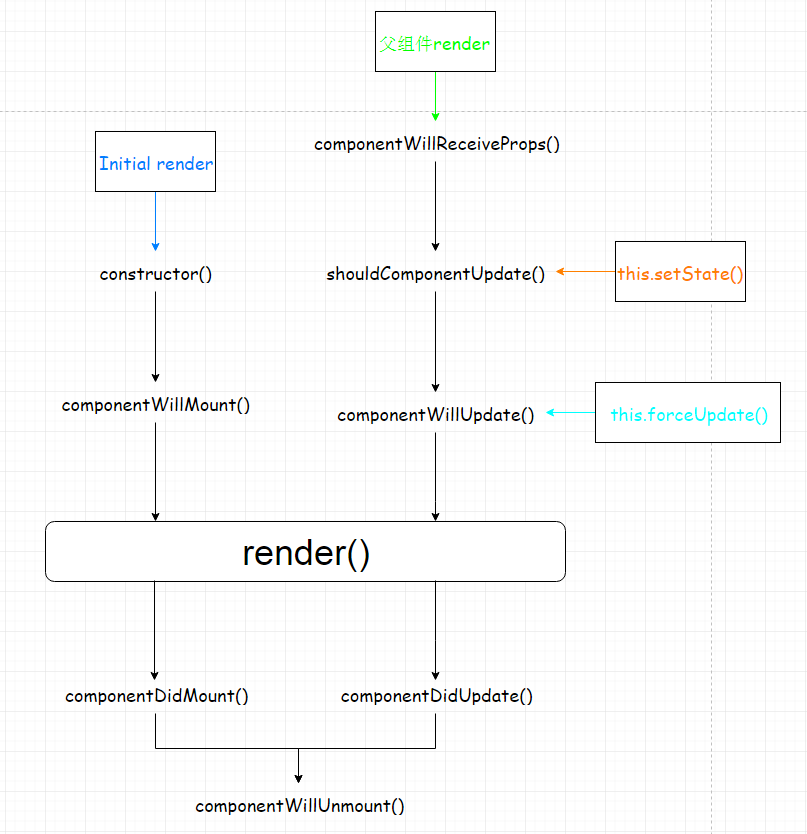
oImg.onload = function() {

obj.src = oImg.src; //下载完成后将该图片赋给目标obj目标对象

}

oImg.src = url; //oImg对象先下载该图像

}



与 POST 相比，GET 更简单也更快，并且在大部分情况下都能用。

然而，在以下情况中，请使用 POST 请求：

* 无法使用缓存文件（更新服务器上的文件或数据库）
* 向服务器发送大量数据（POST 没有数据量限制）
* 发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠
* GET的目的就如同其名字一样是用于获取信息的。它旨在显示出页面上你要阅读的信息。浏览器会缓冲GET请求的执行结果，如果同样的GET请求再次发出，浏览器就会显示缓冲的结果而不是重新运行整个请求。这一流程不同于浏览器的处理过程，但是它是有意设计成这样以使GET调用更有效率。GET调用会检索要显示在页面中的数据，数据不会在服务器上被更改，因此重新请求相同数据的时候会得到相同的结果。
* POST方法应该用于你需要更新服务器信息的地方。如某调用要更改保存在服务器上的数据，而从两个同样的POST调用返回的结果或许会完全不同，因为第二个POST调用的值与第一个的值不相同，这是由于第一个调用已经更新了其中一些值。POST调用通常会从服务器上获取响应而不是保持前一个响应的缓冲。
* 判断js中的数据类型有一下几种方法：typeof、instanceof、 constructor、 prototype、 $.type()/jquery.type()、Object.prototype.toString.call(a) == [Object Array]
* typeof返回的类型都是字符串形式，另外typeof 可以判断function的类型；在判断除Object类型的对象时比较方便。alert(typeof b) ------------> number
* instanceof 后面一定要是对象类型，并且大小写不能错。alert(f instanceof Function) ------------> true
* 通常情况下用typeof 判断就可以了，遇到预知Object类型的情况可以选用instanceof或constructor方法,实在没辙就使用$.type()方法。

tap、touch

一、click 和 tap 比较

两者都会在点击时触发，但是在手机WEB端，click会有 200~300 ms，所以请用tap代替click作为点击事件。

singleTap和doubleTap 分别代表单次点击和双次点击。

三、touch事件touch是针对触屏手机上的触摸事件。现今大多数触屏手机webkit内核提供了touch事件的监听，让开发者可以获取用户触摸屏幕时的一些信息。

其中包括：touchstart,touchmove,touchend,touchcancel 这四个事件

touchstart,touchmove,touchend事件可以类比于mousedown，mouseover，mouseup的触发。

touchstart ： 当手指触摸到屏幕会触发；

touchmove : 当手指在屏幕上移动时，会触发；

touchend : 当手指离开屏幕时，会触发；

而touchcancel许多人不知道它在什么时候会被触发而忽略它，其实当你的手指还没有离开屏幕时，有系统级的操作发生时就会触发touchcancel，例如alert和confirm弹框，又或者是android系统的功能弹窗。

CDN的基本原理是广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对集中的地区或网络中，在用户访问网站时，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，由缓存服务器直接响应用户请求。

link和@import的区别

link属于html标签，而@import是css提供的。

页面被加载时，link会同时被加载，而@import引用的css会等到页面加载结束后加载。（无样式内容闪烁）

link是html标签，因此没有兼容性，而@import只有IE5以上才能识别。

link方式样式的权重高于@import的。

js事件模型

DOM0，DOM2

DOM0：这种事件模型就是直接在dom对象上注册事件名称,而解除事件则更加简单，就是将null复制给事件函数

DOM2：在DOM2级中使用addEventListener和removeEventListener来注册和解除事件（IE8及之前版本不支持）。这种函数较之之前的方法好处是一个dom对象可以注册多个相同类型的事件，不会发生事件的覆盖，会依次的执行各个事件函数。addEventListener('事件名称','事件回调','捕获/冒泡')，首先我们要知道addEventListenr的第一个参数是事件名称，与DOM0级不同的是没有”on“，另外第三个参数代表捕获还是冒泡，true代表捕获事件，false代表冒泡事件。stopPropagation函数用来阻止冒泡。

IE没有事件捕获，event.preventDefault(),阻止默认事件

IE事件模型共有两个过程:

事件处理阶段(target phase)。事件到达目标元素, 触发目标元素的监听函数。

事件冒泡阶段(bubbling phase)。事件从目标元素冒泡到document, 依次检查经过的节点是否绑定了事件监听函数，如果有则执行。

事件绑定监听函数的方式如下:

**attachEvent**(eventType, handler)

事件移除监听函数的方式如下:

**detachEvent**(eventType, handler)

Promise，就是一个对象，用来传递异步操作的消息。有三种状态：Pending（进行中）、Resolved（已完成，又称 Fulfilled）和 Rejected（已失败）。

有了 Promise 对象，就可以将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数。

跨域

CORS：使用自定义的HTTP头部让浏览器和服务器进行沟通

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

Access-Control-Allow-Origin:\*

图像Ping：《img》标签。常用于跟踪用户点击页面或动态广告曝光次数

JSONP：利用动态创建script，src指定一个跨域url

原理是：动态插入script标签，通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。

由于同源策略的限制，XMLHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求。

优点是兼容性好，简单易用，支持浏览器与服务器双向通信。缺点是只支持GET请求。

JSONP：json+padding（内填充），顾名思义，就是把JSON填充到一个盒子里

Comet：长轮询（浏览器定时向服务器发送请求，看有没有数据更新）和流

SSE（服务器发送事件）：这个API用于创建服务器的单向连接，服务器通过这个连接可以发送任意数量的数据。

Web Sockets：在一个单独持久的连接上提供全双工、双向的通信，建立连接后HTTP协议转变为Web Socket协议。 ws:// wss://

通过修改document.domain来跨子域

将子域和主域的document.domain设为同一个主域.前提条件：这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，否则无法利用document.domain进行跨域

主域相同的使用document.domain

使用window.name来进行跨域

window对象有个name属性，该属性有个特征：即在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的，每个页面对window.name都有读写的权限，window.name是持久存在一个窗口载入过的所有页面中的

使用HTML5中新引进的window.postMessage方法来跨域传送数据

还有flash、在服务器上设置代理页面等跨域方式。个人认为window.name的方法既不复杂，也能兼容到几乎所有浏览器，这真是极好的一种跨域方法。

实例对象继承原型对象的constructor指向构造函数

实例对象的proto属性指向原型对象prototype

确定原型和实例的关系

instanceof操作符测试实例和原型链中出现过的构造参数

alert(person1 instanceof Person);//true

isPrototypeOf()方法，只要是原型链中出现的原型，都可以说是该链所派生的实例的原型

alert(Person.prototype.isPrototypeOf(person1));//true

hasOwnProperty()方法，可以检测一个属性是存在于实例中，还是原型中

alert(person1.hasOwnProperty(“name”));实例true，原型false

in操作符，会在通过对象能够访问给定属性时返回true，无论存在于实例还是原型中

alert(“name”in person1);

for-in循环返回所有能够通过对象访问的，可枚举的属性，包括实例和原型中的属性

严格模式

设立”严格模式”的目的，主要有以下几个：

消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;

消除代码运行的一些不安全之处，保证代码运行的安全；

提高编译器效率，增加运行速度；

为未来新版本的Javascript做好铺垫。

注：经过测试IE6,7,8,9均不支持严格模式。

严格模式主要有以下限制：

变量必须声明后再使用

函数的参数不能有同名属性，否则报错

不能使用with语句

不能对只读属性赋值，否则报错

不能使用前缀0表示八进制数，否则报错

不能删除不可删除的属性，否则报错

不能删除变量delete prop，会报错，只能删除属性delete global[prop]

eval不会在它的外层作用域引入变量

eval和arguments不能被重新赋值

arguments不会自动反映函数参数的变化

不能使用arguments.callee

不能使用arguments.caller

禁止this指向全局对象

不能使用fn.caller和fn.arguments获取函数调用的堆栈

增加了保留字（比如protected、static和interface）

说说你对闭包的理解

使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念

闭包有三个特性：

1.函数嵌套函数

2.函数内部可以引用外部的参数和变量

3.参数和变量不会被垃圾回收机制回收

闭包只能取得包含函数中任何变量的最后一个值（for循环中的i，只能取得i最后的值）

ES6的了解

新增模板字符串（为JavaScript提供了简单的字符串插值功能）、箭头函数（操作符左边为输入的参数，而右边则是进行的操作以及返回的值Inputs=>outputs。）、for-of（用来遍历数据—例如数组中的值。）arguments对象可被不定参数和默认参数完美代替。ES6将promise对象纳入规范，提供了原生的Promise对象。增加了let和const命令，用来声明变量。增加了块级作用域。let命令实际上就增加了块级作用域。ES6规定，var命令和function命令声明的全局变量，属于全局对象的属性；let命令、const命令、class命令声明的全局变量，不属于全局对象的属性。。还有就是引入module模块的概念

工厂模式：

主要好处就是可以消除对象间的耦合，通过使用工程方法而不是new关键字。将所有实例化的代码集中在一个位置防止代码重复。

工厂模式解决了重复实例化的问题 ，但还有一个问题,那就是识别问题，因为根本无法搞清楚他们到底是哪个对象的实例。

function createObject(name,age,profession){//集中实例化的函数var obj = new Object();

obj.name = name;

obj.age = age;

obj.profession = profession;

obj.move = function () {

return this.name + ' at ' + this.age + ' engaged in ' + this.profession;

};

return obj;

}

var test1 = createObject('trigkit4',22,'programmer');//第一个实例var test2 = createObject('mike',25,'engineer');//第二个实例

构造函数模式

使用构造函数的方法 ，即解决了重复实例化的问题 ，又解决了对象识别的问题，该模式与工厂模式的不同之处在于：

1.构造函数方法没有显示的创建对象 (new Object());

2.直接将属性和方法赋值给 this 对象;

3.没有 return 语句。

使用构造函数的问题在于每个方法都要在每个实例上重新创建一遍。

原型模式

所有属性被很多实例共享，虽然在实例上添加一个同名属性，可以隐藏原型中的对应属性，但对于包含引用数据类型的属性来说，就有问题了。

构造函数模式用于定义实例属性，原型模式用于定义方法和共有的属性。

js继承方式及其优缺点

原型链继承（本质是重写原型对象，代之以一个新类型的实例）

Sub.prototype=new Super();

缺点：一是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

借用构造函数（类式继承）

借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

组合式继承

组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

说说TCP传输的三次握手四次挥手策略

SYN同步标志、ACK确认标志、FIN结束标志

为了准确无误地把数据送达目标处，TCP协议采用了三次握手策略。用TCP协议把数据包送出去后，TCP不会对传送后的情况置之不理，它一定会向对方确认是否成功送达。握手过程中使用了TCP的标志：SYN和ACK。

发送端首先发送一个带SYN标志的数据包给对方。接收端收到后，回传一个带有SYN/ACK标志的数据包以示传达确认信息。

最后，发送端再回传一个带ACK标志的数据包，代表“握手”结束。

若在握手过程中某个阶段莫名中断，TCP协议会再次以相同的顺序发送相同的数据包。

断开一个TCP连接则需要“四次握手”：

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可 以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。

第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。

第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

说说你对作用域链的理解

作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到window对象即被终止，作用域链向下访问变量是不被允许的。

HTTP和HTTPS

HTTP协议通常承载于TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TLS），这个时候，就成了我们常说的HTTPS。

默认HTTP的端口号为80，HTTPS的端口号为443。

为什么HTTPS安全

因为网络请求需要中间有很多的服务器路由器的转发。中间的节点都可能篡改信息，而如果使用HTTPS，密钥在你和终点站才有。https之所以比http安全，是因为他利用ssl/tls协议传输。它包含证书，卸载，流量转发，负载均衡，页面适配，浏览器适配，refer传递等。保障了传输过程的安全性

cookie

Cookie 是在 HTTP 协议下，服务器或脚本可以维护客户工作站上信息的一种方式。Cookie 是由 Web 服务器保存在用户浏览器（客户端）上的小文本文件，它可以包含有关用户的信息。无论何时用户链接到服务器，Web 站点都可以访问 Cookie 信息

1.`Cookie`数量和长度的限制。每个domain最多只能有20条cookie，每个cookie长度不能超过4KB，否则会被截掉.

2.安全性问题。如果cookie被人拦截了，那人就可以取得所有的session信息。即使加密也与事无补，因为拦截者并不需要知道cookie的意义，他只要原样转发cookie就可以达到目的了。

3.有些状态不可能保存在客户端。例如，为了防止重复提交表单，我们需要在服务器端保存一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗

考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能

考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议：

将登陆信息等重要信息存放为SESSION

其他信息如果需要保留，可以放在COOKIE中

内存泄漏

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。

垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收。

setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。

闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

列举IE 与其他浏览器不一样的特性？

IE支持currentStyle，FIrefox使用getComputStyle

IE 使用innerText，Firefox使用textContent

滤镜方面：IE:filter:alpha(opacity= num)；Firefox：-moz-opacity:num

事件方面：IE：attachEvent：火狐是addEventListener

鼠标位置：IE是event.clientX；火狐是event.pageX

IE使用event.srcElement；Firefox使用event.target

IE中消除list的原点仅需margin:0即可达到最终效果；FIrefox需要设置margin:0;padding:0以及list-style:none

CSS圆角：ie7以下不支持圆角

attribute和property的区别是什么？

attribute是dom元素在文档中作为html标签拥有的属性；

property就是dom元素在js中作为对象拥有的属性。

所以：对于html的标准属性来说，attribute和property是同步的，是会自动更新的，但是对于自定义的属性来说，他们是不同步的

Etag

聪明的服务器开发者会把ETags和GET请求的“If-None-Match”头一起使用，这样可利用客户端（例如浏览器）的缓存。因为服务器首先产生ETag，服务器可在稍后使用它来判断页面是否已经被修改。本质上，客户端通过将该记号传回服务器要求服务器验证其（客户端）缓存。

其过程如下：

客户端请求一个页面（A）。 服务器返回页面A，并在给A加上一个ETag。 客户端展现该页面，并将页面连同ETag一起缓存。 客户再次请求页面A，并将上次请求时服务器返回的ETag一起传递给服务器。 服务器检查该ETag，并判断出该页面自上次客户端请求之后还未被修改，直接返回响应304（未修改——Not Modified）和一个空的响应体。

私有变量

包括函数的参数、局部变量和在函数内部定义的其他函数

特权方法

1.在构造函数中定义特权方法（实例变量）

function(){

//私有属性和私有函数

var privateVariable=10;

function privateFunction(){

return false;

}

//特权方法

this.publicMethod=function(){

privateVariable++;

return privateFunction();

};

}

2.在私有作用域(立即执行匿名函数)中定义私有变量和函数，同样也可以创建特权方法（静态私有变量）

(function(){

//私有属性和私有函数

var privateVariable=10;

function privateFunction(){

return false;

}

//构造函数（函数声明只能创建局部函数，出于同样原因，未使用var）

MyObject=function(){

};

//公有、特权方法

MyObject.prototype.publicMethod=function(){

privateVariable++;

return privateFunction();

};

})();

BFC（块级格式化上下文）就是一个相对独立的布局环境，它内部元素的布局不受外面布局的影响。它可以通过以下任何一种方式来创建：

float 的值不为 none

position 的值不为 static 或者 relative

display 的值为 table-cell , table-caption , inline-block , flex , 或者 inline-flex 中的其中一个

overflow 的值不为 visible

约束规则：

1、内部的Box会在垂直方向上一个接一个的放置

2、垂直方向上的距离由margin决定。（完整的说法是：属于同一个BFC的两个相邻Box的margin会发生重叠，与方向无关。）

3、每个元素的左外边距与包含块的左边界相接触（从左向右），即使浮动元素也是如此。（这说明BFC中子元素不会超出他的包含块，而position为absolute的元素可以超出他的包含块边界）

4、BFC的区域不会与float的元素区域重叠

5、计算BFC的高度时，浮动子元素也参与计算

6、BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面元素，反之亦然

页面的加载顺序

html顺序加载，其中js会阻塞后续dom和资源的加载，css不会阻塞dom和资源的加载但是会阻塞js的加载。

浏览器会使用prefetch对引用的资源提前下载

1.没有 defer 或 async，浏览器会立即加载并执行指定的脚本

2.有 async，加载和渲染后续文档元素的过程将和 script.js 的加载与执行并行进行(下载异步，执行同步，加载完就执行)。

3.有 defer，加载后续文档元素的过程将和 script.js 的加载并行进行（异步），但是 script.js 的执行要在所有元素解析完成之后，DOMContentLoaded 事件触发之前完成。

二叉树的遍历的三种方式

//(1)前序遍历（DLR）

function preOrder(node){

if(node){

arr.push(node);

preOrder(node.firstElementChild);

preOrder(node.lastElementChild);

}

}

//(2)中序遍历（LDR）

function inOrder(node){

if(node){

inOrder(node.firstElementChild);

arr.push(node);

inOrder(node.lastElementChild);

}

}

//(3)后序遍历（LRD）

function postOrder(node){

if(node){

postOrder(node.firstElementChild);

postOrder(node.lastElementChild);

arr.push(node);

}

}

中间件

中间件就是一个函数，对store.dispatch方法进行了改造，在发出 Action 和执行 Reducer 这两步之间，添加了其他功能。

将它放在applyMiddleware方法之中，传入createStore方法，就完成了store.dispatch()的功能增强。

createStore方法可以接受整个应用的初始状态作为参数，那样的话，applyMiddleware就是第三个参数了。

applyMiddleware方法的三个参数，就是三个中间件。有的中间件有次序要求，使用前要查一下文档。比如，logger就一定要放在最后，否则输出结果会不正确。

applyMiddlewares()是 Redux 的原生方法，作用是将所有中间件组成一个数组，依次执行。

异步操作的基本思路

同步操作只要发出一种 Action 即可，异步操作的差别是它要发出三种 Action。

操作开始时，送出一个 Action，触发 State 更新为"正在操作"状态，View 重新渲染

操作结束后，再送出一个 Action，触发 State 更新为"操作结束"状态，View 再一次重新渲染

connect方法接受两个参数：mapStateToProps和mapDispatchToProps。它们定义了 UI 组件的业务逻辑。前者负责输入逻辑，即将state映射到 UI 组件的参数（props），后者负责输出逻辑，即将用户对 UI 组件的操作映射成 Action。

mapStateToProps()

mapStateToProps会订阅 Store，每当state更新的时候，就会自动执行，重新计算 UI 组件的参数，从而触发 UI 组件的重新渲染。

mapStateToProps的第一个参数总是state对象，还可以使用第二个参数，代表容器组件的props对象。

connect方法可以省略mapStateToProps参数，那样的话，UI 组件就不会订阅Store，就是说 Store 的更新不会引起 UI 组件的更新。

mapDispatchToProps()

mapDispatchToProps是connect函数的第二个参数，用来建立 UI 组件的参数到store.dispatch方法的映射。也就是说，它定义了哪些用户的操作应该当作 Action，传给 Store。它可以是一个函数，也可以是一个对象。